Benchmarking Mixture-of-Recursion (MoR) on Fine-tuned Small Language Model

Sumaiya Sultana
Dept. of Computer Science & Engineering
Feni University
Feni, Bangladesh
ID: 213031009

Shazzad Hossain Mazumder
Dept. of Computer Science & Engineering
Feni University
Feni, Bangladesh
Supervisor

***Abstract*—This paper investigates Mixture-of-Recursion (MoR), a novel architecture that dynamically adapts computational depth to enhance efficiency and performance in language modeling. By allowing each layer to recursively re-apply itself, execute once, or be skipped based on input complexity, MoR enables dynamic depth in sequence modeling. Unlike standard Transformers that mandate a fixed number of layers for every token, MoR employs lightweight routing networks to adaptively control recursive computation. We benchmark MoR-based models against standard fixed-depth Transformers using a character-level dataset derived from Tiny Shakespeare for next-token prediction tasks. Two primary experiments are conducted: an efficiency profiling experiment where a 12-layer MoR Transformer is fine-tuned with an auxiliary recursion penalty to reduce effective depth, and an equivalent-cost experiment where a 12-layer MoR model is fine-tuned to operate at an effective depth comparable to a 6-layer baseline. Results demonstrate that MoR consistently outperforms standard Transformers. In the efficiency experiment, the MoR-Transformer achieves a 33% reduction in effective depth while maintaining comparable accuracy. In the equivalent-cost experiment, the MoR model demonstrates significantly superior generalization, achieving higher accuracy on held-out data than the 6-layer baseline at a similar computational cost. These findings highlight MoR as a promising approach for constructing compute-efficient Transformer models that optimize performance by adaptively allocating computational resources.**

***Index Terms*—Mixture-of-Recursion (MoR), LLM, SLM, Transformer, Recursive wise KV cache, Routing Mechanism.**

I. Introduction

A. Background of The Research

Large Language Models (LLMs) built on the Transformer architecture have established themselves as the cornerstone of modern artificial intelligence, demonstrating exceptional performance in tasks such as text generation, code synthesis, and question answering [1]–[3]. The efficacy of Transformers stems from their capacity to process sequences in parallel and capture complex patterns within extensive datasets.

However, the standard Transformer architecture suffers from significant limitations. Primarily, it imposes a fixed computational depth across all inputs, disregarding the inherent variability in token complexity. Consequently, simple tokens receive the same computational resources as complex reasoning phrases, leading to inefficient resource allocation. Furthermore, the quadratic scaling of the self-attention mechanism with respect to sequence length results in substantial memory and computational overhead [4]–[6]. These factors collectively render large models expensive to train and challenging to deploy.

This has motivated growing interest in Small Language Models (SLMs), which aim to achieve competitive performance using fewer parameters and lower computational budgets. SLMs are particularly important for deployment in resource-constrained environments such as edge devices, mobile platforms, and real-time systems [7]. However, because SLMs have limited capacity, they must utilize computation more efficiently. This makes adaptive computation mechanisms especially critical for SLM-based architectures.

Recent studies indicate that next-token prediction facilitates the learning of critical linguistic aspects, including grammar, semantics, and reasoning patterns [8], [9]. Yet, the complexity of predicting the subsequent token varies significantly. While some tokens are predictable via local context or frequency patterns, others necessitate long-range context, multi-step reasoning, or recursive processing.

To mitigate these issues, dynamic computation paradigms have emerged, enabling models to adjust layer usage based on input complexity. Mixture-of-Experts (MoE) exemplifies this approach by routing tokens to specific expert layers to conserve computation [10]. However, MoE is limited to spatial routing and lacks the capability for temporal recursion, meaning it cannot re-apply a layer to deepen reasoning.

The Mixture-of-Recursions (MoR) architecture addresses this limitation by offering each layer three execution choices: skip, execute once, or recurse. This flexibility allows for adaptive recursion, where complex tokens receive increased computation [11]. Such adaptability is vital for tasks with variable difficulty, such as code generation and question answering, positioning MoR as a potent architecture for enhancing both efficiency and reasoning capabilities.

B. Problem Definition

Although Transformers are powerful, but their fixed-depth structure creates several challenges:

- **Computational inefficiency:** Every input passes through the full layer stack, even when not necessary [12].

- **Lack of recursive computation:** Existing dynamic-routing methods like MoE cannot re-visit a layer, meaning the model cannot naturally perform deeper iterative reasoning [10].
- **Scaling bottlenecks:** Larger models demand huge memory and compute budgets, making them impractical for many applications. Continuously increasing model size to gain performance is economically and environmentally unsustainable. There is a strong need for smarter and more efficient architecture that uses less parameters.
- **Limited benchmarking of MoR:** While MoR has shown benefits on general language modeling, its performance on tasks that require deeper processing remains underexplored [11].

These limitations raise important questions:

- Can MoR reduce computational cost while maintaining accuracy?
- Does MoR perform better than a fixed-depth Transformer when both use the same amount of computation?
- How does recursion frequency correlate with input difficulty in next token prediction tasks?

To address these inquiries, this study conducts a comparative analysis of MoR and standard Transformers.

C. Research Aim and Objectives

The primary objective of this research is to evaluate the efficacy of the Mixture of Recursions (MoR) architecture as an adaptive computation method for next-token prediction. Furthermore, we aim to benchmark MoR against standard fixed-depth Transformers to ascertain which approach offers superior efficiency and performance.

To achieve this aim, the research focuses on the following objectives:

- Benchmark and Evaluate Performance: Measure how well MoR and standard Transformers perform looking at accuracy and effective depth—using small-scale models.
- Test MoR's Efficiency: Examine whether MoR can use less computation (lower effective depth) while still keeping the same or better performance.
- Comparison: Compare MoR and fixed-depth Transformers under the same computational budget for next token prediction tasks.
- Understand MoR's Behavior: Study MoR's internal routing, such as recursion rates, skip rates, and execution patterns, to see how the model adjusts its computation based on task difficulty.

II. Literature Review

A. Introduction

Large Language Models (LLMs) represent a breakthrough in artificial intelligence, enabling machines to understand, generate, and process human-like text for tasks like answering questions, code generation, summarizing, and translating. LLM built on transformer architecture [13]. Mixture-of-Recursions (MoR) is a new model using parameter sharing, adaptive computation, and a routing mechanism to enhance the efficiency of language model.

This chapter presents reviews of existing studies related to Large Language Models (LLMs), Small Language Models (SLMs), transformer architectures, Recursive Transformer, the Mixture of Expert (MoE) and the Mixture-of-Recursions (MoR) framework. The goal is to identify the strengths and weaknesses of current models so that we will increase the effectiveness of our model.

B. Large Language Model (LLM)

Large language model (LLM) is a language model trained on a vast amount of text, usually based on the Transformer architecture. It designed for natural language processing tasks, especially for language generation. By scaling parameters to billions and using self-supervised learning, they acquire versatile capabilities in understanding and generating human like text across many tasks. The development of large language models (LLMs) began with GPT-1 the first model to apply generative pre-training to a decoder-only transformer [14].

While GPT-1 introduced generative pre-training, the another model GPT-3 a 175-billion parameter model demonstrated that sufficiently large models could learn to follow instructions and solve diverse tasks with minimal examples, no special training needed [2]. However, the empirical scaling approach of these early models lacked a theoretical foundation for optimal resource allocation, that was systematically addressed by Kaplan et al. [15], who formulated scaling laws, showing power-law relationships between model size, dataset size, training compute, and performance. By Following the scaling law Later models did even better. Google's PaLM used 540 billion parameters scaled across different computers easily, it demonstrated breakthrough performance on reasoning tasks, achieving human-level performance [16]. GPT-4 beats human experts on many tests, it processes both text and image inputs to achieve human-level performance [17]. Though these LLM models have significant advancement and achieve great performance but these models give every word the full model depth. This wastes power on simple words like "the".

C. Small Language Model (SLM)

Small Language Model (SLM) is a Transformer based AI generative Model which is similar to Large Language Model (LLM) but with a significantly reduced size. Small Language Models (SLMs) are lightweight versions of traditional language models. Typically, large language models have 100 billion to 10+ trillion parameters, On the other hand, SLMs have 1 million to 10 billion parameters. Though small language models are significantly smaller then LLM but they still retain core NLP capabilities [18]. Large language models (LLMs) exhibit exceptional comprehension and reasoning capabilities across a wide range of tasks. LLMs offer maximal general capability at high cost, while SLMs emphasize efficiency, privacy, and specialization.

However, the massive size, high cost and complexity of LLMs can pose significant challenges for some applications

and make them impractical for many use cases. As a solution to these problems small language models (SLMs) were invented [19]. The Aim of SLM is to make machine intelligence more accessible, affordable, and efficient for everyday tasks [20]. The SLM journey began in 2019 with the DistilBERT model. It compressed a LLM model - BERT from 110M to 66M parameter and achieved 97% of BERT performance at 60% faster inference. Which proved that small model could retain capability [21]. The Modern SLM era was explored in 2023 with Microsoft Phi family. At 1.3 billion parameters Phi-1 achieves coding performance competitive with models over 100× larger [22]. Phi-2 achieved 58% MMLU (Massive Multitask Language Understanding) beating 25B LLMs [23]. Phi-3 Mini with 3.8 billion parameters reached 68.8% MMLU beats Llama 70B and running on smartphones [24]. This evolution of SLM makes machine intelligence accessible, affordable, and efficient for everyday tasks and it proves that capability doesn't require trillions of parameters.

However there remains a problem that all model uses the same number of layers for every word because SLM a Transformer based model. This Fixed-depth gap creates the MoR opportunity because MoR use dynamic recursive depths.

D. Transformer

The transformer architecture proposed by Vaswani et al. [12] introduces a mechanism known as "self-attention," which replaces recurrence and sequence modeling, allowing for parallel processing. This self-attention mechanism has dramatically improved training efficiency and better handled long-range dependencies in text compared to RNNs/LSTMs. Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence [12].

Transformers have layered encoder-decoder blocks that hold multi-head attention and feed-forward sublayers. The decoder includes an extra attention layer that considers outputs from the encoder stack. The transformer uses positional encoding, allowing the entire sequence to be processed in parallel and ensuring that word order information is preserved [12].

Large Language Models (LLMs) have brought about a significant revolution in the world of language models, and the Transformer architecture serves as the core foundation of LLMs. The transformer has become the backbone of almost all large language models (LLMs), such as BERT, GPT, and T5. However, transformers have enabled significant advancements in tasks like code generation, code summarization, complex reasoning and QA, but despite these successes, this architecture has several limitations, such as high computational cost, memory bottlenecks, and no adaptability. Bae et al. [25] introduce Relaxed Recursive Transformers that use parameter sharing in large language models (LLMs) to reduce size and cost. This model reuses layers with minimal performance loss. But the MoR (Mixture-of-Recursion) combine parameter sharing and adaptive computation, the two axes of efficiency inside a single Recursive Transformer [11]. This can overcome all the limitations of previous transformer architectures.

E. Recursive Transformer

Recursive transformer" usually refers to adding explicit recursion, iteration, or recursive structure on top of standard transformer blocks to improve efficiency or enable algorithmic/generalization behavior. It emerged as a solution of "fixed computational depth for all inputs", which is the fundamental limitation of standard transformers. The Universal Transformer introduced recurrence and adaptive computation time, addressing the fixed-depth limitation of standard transformers [26]. Universal Transformers don't work perfectly in practice, it works well only on short sequences and it applies the same amount of computation to every word in a sentence. This is where Mixture-of-Recursions (MoR) comes in. Think of MoR as a smart upgrade to the Universal Transformer. It adds a "router" that looks at each word individually [11].

F. Mixture of Expert

Mixture of Expert (MoE) is a neural network architecture that was designed to improve model capacity and computational efficiency. It used many specialized subnetworks ("experts") which share the work and a gating network dynamically selects which experts to activate for each input. It has become central to scaling modern large language models (LLMs) and other deep learning systems while keeping per-token compute roughly constant [27], [28]. The idea of Mixture of Experts (MoE) originates from the work of Jacobs et al. [29] where different 'experts' handled different parts of a problem.

Shazeer et al. [10] introduces the modern application of MoE. This paper introduce the Sparsely-Gated Mixture-of-Experts (MoE) layer, a new neural network component designed to achieve conditional computation. It used top-k gating to select a small subset of experts per token. But this model have training instability due to discrete routing. Switch Transformers Reduced this instability issues, it simplified the original MoE design by routing each token to only one expert instead of topk [30]. Where Switch transformer used one expert per token, in another study Jiang et al. [31] used two experts per token and Improved Balance and Efficiency. The another model DeepSeek-V2 introduced routing diversity with MLA and fine-grained MoE [32]. The new model MoR mix routing strategies to optimize efficiency, specialization, and generalization.

G. Mixture of Recursion

Mixture-of-Recursion (MoR) proposed by Bae et al. [11], a unified framework that combines parameter sharing and adaptive computation inside a single Recursive Transformer to enhance the efficiency of large language models (LLMs). In Traditional Transformers [12], each input goes through the same fixed number of layers, and the model always does the same amount of work. On the other hand, MoR reuses a single set of shared layers across multiple recursion steps and it allows each token to be processed based on how difficult the task is. MoR doesn't hold memory like Transformer.

Parameter sharing techniques reduce the total number of unique parameters by reusing weights, which reduce the overall computational complexity. Parameter sharing techniques build on the concept of Recursive Architecture. Sharing weights across layers, enabling extremely deep networks with minimum parameter growth and computational cost [25], [26], [33]. Sliced Recursive Transformer achieve 10-30% less FLOPs by using parameter sharing mechanism [33]. Adaptive computation makes computation adaptive, not fixed. Each token can decide how many recurrent steps its need, easy token-fewer steps, complex token-more steps [26]. Elbayad et al. [34] proposed a depth-adaptive transformer that uses a halting mechanism to dynamically skip layers during inference. This model can achieve computation savings of 20-40%. But it's makes training more complex and less stable. MoR uses a lightweight router that assigns a specific recursion depth to each token. This concept is similar to the routing mechanism in Mixture of Expert (MoE) models, where a router directs tokens to specialized expert networks. In this paper MoE achieves Lower perplexity at scale and 4x faster efficiency [10]. In another study, which also uses routing mechanism archives better stability over Traditional MoE, it uses top-1 routing where MoE uses top-2 routing [30]. Relaxed Recursive Transformer use parameter sharing and adaptive computation(depth-wise batching) gains 2-3x speedup [25].

MoR models use parameter sharing, adaptive computation and also routing mechanisms. This model achieves 5-10% Perplexity, save 20-30% Flops, have two times faster inference, need 50% smaller memory, better performance than much bigger transformer models [11].

III. Methodology

This chapter details the methodology employed to design, implement, train, and evaluate the proposed Mixture of Recursions (MoR) Transformer architecture. The objective is to empirically assess whether MoR can dynamically adapt computational depth while maintaining or enhancing performance relative to standard fixed-depth interactions. We compare MoR Transformer efficiency against standard transformers on a character-level language modeling task, drawing design inspiration from recent advancements in MoR that introduce dynamic recursive depths for adaptive token-level computation.

A. Dataset and Preprocessing

1) Dataset: The dataset built from a Tiny Shakespeare excerpt (Romeo & Juliet balcony scene, ∼1,200 characters) is used as a controlled language modeling benchmark. We used this dataset for the next token predictions task. It is taken from Andrej Karpathy charRNN project [35]. The dataset is also available in TensorFlow Datasets and the Hugging Face Hub. The dataset is chosen because -It has rich linguistic structure; it is small enough for rapid experimentation and it enables reproducibility. Vocab Size is 55 characters. Generated 5,000 sequences of length 64 for causal language modeling (next-token prediction).

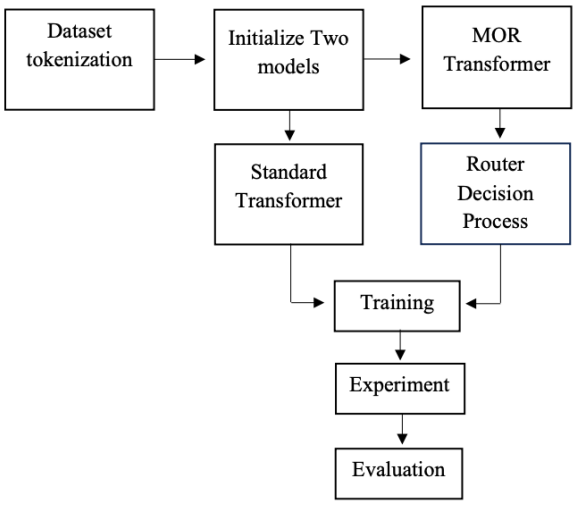

Fig. 1. Proposed Method

2) Tokenization: Character-level tokenization is applied. Each character is mapped to a unique integer. All unique characters in the corpus (including spaces, punctuation, and line breaks) are collected into a sorted set. Two lookup tables are defined in Table I.

TABLE I
Tokenization

Name	Description
stoi	maps each character to a unique integer ID
itos	inverse mapping from ID back to character

The full corpus is then encoded as a one-dimensional tensor of integer token IDs.

B. Research Pipeline

The research pipeline, illustrated in Figure 1, operates sequentially, beginning with dataset tokenization to prepare the input data. This is followed by the initialization of two distinct models: the MoR Transformer and the Standard Transformer. The process then incorporates the router decision mechanism before proceeding to the fine-tuning phase. Finally, comprehensive experiments are conducted, concluding with a rigorous evaluation of the models' performance.

C. Model Architecture

1) Baseline Transformer: The baseline model is a standard Transformer encoder-decoder based language model consisting of: - Token embedding layer, Positional encoding, A stack of fixed Transformer layers, Output projection layer for next-token prediction. Each Transformer layer includes: multi-head self-attention, Feed-forward neural network, Residual connections and layer normalization. This baseline executes all layers for every input, resulting in a fixed computational depth equal to the number of layers N [12].

The base Transformer configuration is shown in Table II.

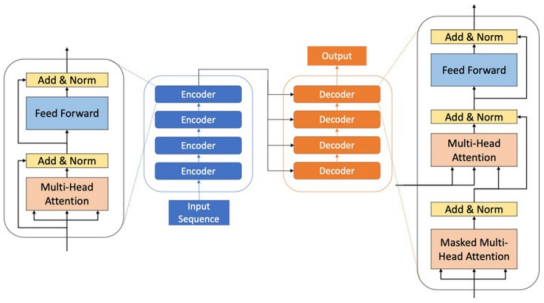

Fig. 2. Transformer Architecture Diagram

TABLE II
BASE TRANSFORMER CONFIGURATION

Configuration	Description
Embedding dimension	$d_{model} = 256$
Number of layers	$N_{full} = 12$ (deep configuration) $N_{low} = 6$ (shallower baseline).
Attention	Multi-head self-attention with 8 heads Batch-first layout Inputs are shaped $B \times L \times d_{model}$
Feedforward network	Hidden dimension: $d_{ff} = 2048$. Activation: ReLU.
Normalization	LayerNorm after attention and feedforward sublayers
Dropout probability	0.1
Positional encoding	Standard sinusoidal positional encoding with maximum length 5000, truncated to the sequence length $L = 64$

2) MoR-Transformer: MoR-transformer use same backbone as baseline transformer, it extends the standard Transformer by introducing a routing mechanism that enables conditional execution, skipping, or recursion at each layer. Each MoR Transformer layer optionally includes: A standard Transformer computation block, An MoR Router module. The router decides per input sample whether to: Skip the layer, Execute the layer once, recurse by re-applying the same layer multiple times. This design allows the model to allocate more computation to complex inputs and less computation to simpler inputs, enabling adaptive depth.

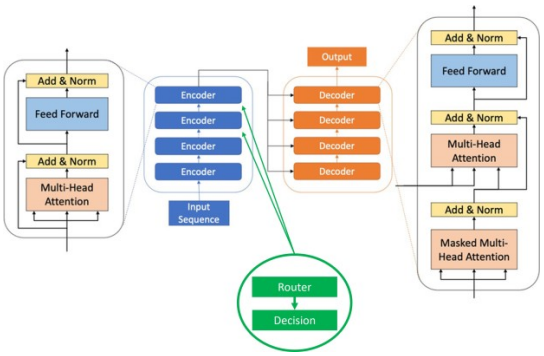

Fig. 3. MoR-Transformer Diagram

MoR use Parameter Sharing, Adaptive Computation, Recursion Wise KV cache and Routing Mechanism for enhance the efficiency of Language Model [11].

Parameter Sharing: Parameter sharing is the process of reusing a single set of shared layers across multiple recursion steps to reduce the total number of unique parameters. This makes a model more efficient by using fewer parameters [11] [33].

Adaptive Computation: MoR uses lightweight routers trained end-to-end to dynamically assign a token-specific recursion depth. This dynamic allocation ensures that computational effort is focused only on tokens that are still active at a given recursion depth, effectively directing "thinking" to where it is most needed.

Recursion Wise KV cache: Dynamic recursion routing only cache what is actually used at each recursion, so KV memory and bandwidth drop while throughput improves. At each recursion step, only the tokens that are routed to that step store their keys and values in the cache for that level.

MoR Routing Mechanism: The MoR-Router is the core of the adaptive mechanism. The router uses a multi-layer perceptron (MLP) followed by a SoftMax layer to output probabilities over three routing actions:

- Recurse (Cost:1+): execute the layer and then dynamically call the same layer again (up to a maximum recursion depth).
- Forward/Execute (Cost:1): execute the layer once and continue.
- Skip (Cost:0): bypass the layer entirely.

During training, Gumbel-SoftMax is applied to maintain differentiability, while during inference, hard argmax decisions are used.

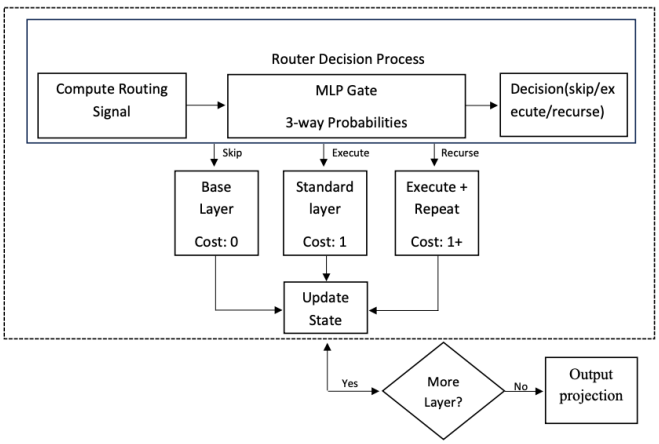

Fig. 4. Routing Mechanism

D. Experiments

We conducted two experiments:

- **Experiment 1 (Efficiency Profiling):** Compares a 12-layer baseline Transformer with a 12-layer MoR-Transformer trained on identical data and epochs.
- **Experiment 2 (Equal Cost Comparison):** Compares a 6-layer baseline Transformer against a 12-layer MoR

model trained to effectively utilize approximately 6 layers (E $\approx$ 6).

Split data: Experiment 1 use the full set as training/evaluation. Experiment 2 use first half for training (simple/seen), second half as held-out test (complex/unseen).

E. Fine-tuning Process

Both models are fine-tuned as language models on the Tiny Shakespeare excerpt dataset. The fine-tuning pipeline followed these configurations in Table III.

TABLE III
FINE-TUNING CONFIGURATION

Parameter	Value/Description
Batch size	64
Architecture	Transformer/Mor-Transformer
d_model(hidden size)	256
Num. layers (N_{full})	12
Num. layers baseline low (N_{low})	6 (for equal-compute experiment)
Optimizer	AdamW (base params), AdamW (router params)
learning rate	1e-3
Baseline (all exp)	30 epochs
MoR (Exp 1)	30 epochs
MoR (Exp 2)	50 epochs (to optimize routing)
Platform and Hardware	Kaggle, Gpu

F. Evaluation metrics

To understand how well our models are working, we used some standard evaluation metrics. These help us measure accuracy and speed of our model.

1) Effective Depth (E): Effective Depth (E) is the average number of layers actually executed by a model during inference or training. It is a measure of the realized computational depth of a dynamically-routed or recursive neural architecture.

2) Accuracy: Accuracy measures top-1 next-token prediction correctness for language modeling. It shows what fraction of predicted tokens match true next tokens.

3) Held-out accuracy: Held-out accuracy is the classification accuracy measured on data that was not used for training (the held-out or test set). It tells how well the model generalizes to unseen examples.

4) Fine-tuning Time: Fine-tuning time is the total amount of time a model spends to finish fine-tuning on a given dataset. It helps evaluate: Efficiency (how fast a model trains), Comparison between models (e.g., Standard Transformer vs. MoR), Compute cost and Time-to-convergence.

IV. RESULT AND ANALYSIS

This chapter presents the experimental results that were obtained after training and evaluating our proposed model. The results are analyzed based on metrics such as accuracy, effective depth, train time. The performance of the MoR-Transformer model is discussed in detail to demonstrate its effectiveness.

A. Results

This section details the results of the Efficiency Profiling and Equal Cost Comparison experiments.

1) Experiment 1: Efficiency Profiling (N=12 vs MoR N=12): Goal: Compare a normal Transformer (12 layers executed always) with a MoR Transformer (12 recursive layers allowed to skip/recurse dynamically).

TABLE IV
EXPERIMENT 1 RESULT

Metric	Standard (N=12)	MoR (N=12)
Accuracy	22.7634	22.7635
Effective Depth (E)	12.00	8.00
Fine-tuning Time	108 s	83 s

Interpretation: The results indicate that the MoR model achieves virtually identical predictive accuracy to the standard Transformer (approx. 22.76). Notably, the MoR router utilized an average of only 8 layers per sample, compared to the fixed 12 layers of the baseline. This 33% reduction in computations per token sequence underscores the efficiency gains of adaptive routing, leading to faster and more resource-efficient training without compromising accuracy.

2) Experiment 2: Performance Under Equivalent Cost (N=6 vs MoR N=12, E $\approx$ 6): Goal: See if a MoR model (12 potential layers, dynamic depth $\approx$ 6) can match the compute cost of a smaller baseline (6 layers) while achieving better accuracy through adaptive computation.

TABLE V
EXPERIMENT 2 RESULT

Metric	Standard (N=6)	MoR (N=12, E$\approx$6)
Held-out Accuracy	39.87 %	49.67 %
Effective Depth (E)	6.00	5.89
Fine-tuning Time	30 s	59 s

Interpretation: With an effective depth E $\approx$ 6, the MoR model operated within the same average compute budget as the 6-layer baseline. Despite equivalent computational cost, the MoR model achieved a 10% absolute improvement in held-out accuracy, demonstrating superior generalization. The router effectively learned to deepen recursion for complex inputs while skipping layers for simpler ones, enabling the model to function as a variable-depth expert system that efficiently balances computation and accuracy.

TABLE VI
OVERALL OUTCOME

Experiment	What it proved	Key Takeaway
1: Efficiency	For same Layer (12 L), MoR reduces compute (E = 8) with identical accuracy.	MoR learns to skip unnecessary computation.
2: Equivalent Cost	For same average cost (E $\approx$ 6), MoR (12 L) achieves higher accuracy than fixed 6 L.	MoR uses depth where needed $\rightarrow$ better adaptability and generalization.

3) Overall Experimental Outcome:

B. Result Analysis

In Experiment 1, the MoR-Transformer reduced effective depth by 33% (E=8.00 vs baseline E=12.00) while keeping

accuracy almost the same (22.7635 vs 22.7634). This shows that the routing mechanism can learn to skip unnecessary layers and control recursion under a strong auxiliary penalty ($\lambda = 0.1$), which speeds up training by 23% (83s vs 108s) without performance degradation.

Experiment 2 revealed even stronger benefits: on held-out data, the MoR model (N=12, E=5.89) outperformed the shallower baseline (N=6, E=6.00) by 24.6% in accuracy (49.6687 vs 39.8700) at marginally lower computational cost, confirming that MoR leverages its greater nominal capacity through adaptive execution paths—allocating deeper recursion to complex sequences while skipping simpler ones—thus validating both core hypotheses that MoR provides superior efficiency at equal depth and better generalization under equivalent compute budgets, consistent with broader MoR literature reporting 2x inference gains through dynamic recursion.

V. Conclusion & Future Work

A. Conclusion

This thesis explored the effectiveness of the Mixture of Recursions (MoR) architecture as an adaptive computation mechanism for next-token prediction tasks, with a particular focus on small language models (SLMs). Unlike standard Transformers, which apply a fixed depth of computation to every input, MoR introduces a lightweight routing mechanism that allows each token to either skip, execute once, or recurse through a shared layer. This design enables the model to allocate more computation to complex tokens while saving resources on simpler ones.

Two experiments were conducted using a character-level dataset inspired by Tiny Shakespeare. The Experimental comparisons between MoR and standard Transformers demonstrate that MoR can achieve superior performance while using fewer effective layers. This model not only lowers effective depth but also preserves and improves accuracy compared to standard Transformers under the same computational budget. The study confirms MoR supports recursive reasoning, which is absent in traditional MoE models, and adapts its routing behavior based on token complexity. These capabilities make MoR a scalable and efficient solution for language modeling, especially in resource-constrained environments.

In conclusion, our findings position MoR as a viable alternative to fixed-depth Transformers, particularly in resource-constrained scenarios favoring SLMs, such as mobile and edge computing. By enhancing computational efficiency without sacrificing accuracy, MoR represents a significant step towards sustainable and scalable language modeling.

B. Limitation

However, while MoR shows clear advantages, certain trade-offs remain. Recursive mechanisms introduce training instability and tuning complexity. Real-world deployment metrics such as inference latency, throughput, and energy consumption on GPUs/TPUs were not profiled. Routing decisions relied on a simple controller with fixed auxiliary loss weight. More adaptive or learned routing strategies were not explored.

C. Future Works

Although this research provides strong evidence for the benefits of adaptive recursion, several avenues remain open for future investigation. One important extension is to evaluate MoR on larger and more diverse datasets, including code generation, question answering, translation and summarization to better understand its generalization ability. Combine MoR with MoE to leverage both expert specialization and recursive depth control for enhanced performance. Finally, deploying MoR on resource constrained hardware like smartphones or IoT devices would demonstrate its practical benefits for realtime systems, and exploring its extension to multimodal tasks (text plus image or text plus code) could open new opportunities for adaptive recursion in broader domains.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] T. Brown, B. Mann, N. Ryder, et al. Language models are few-shot learners. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 33, pages 1877–1901, 2020.

[3] C. Raffel, N. Shazeer, A. Roberts, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1):5485–5551, 2020.

[4] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proc. 57th Annu. Meet. Assoc. Comput. Linguist. (ACL)*, pages 2978–2988, 2019.

[5] N. Kitaev, L. Kaiser, and A. Levskaya. Reformer: The efficient transformer. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[6] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[7] L. Ben Allal et al. SmolLM2: When smol goes big–data-centric training of a small language model. *arXiv preprint*, 2025.

[8] Y. Li, Y. Huang, M. E. Ildiz, A. S. Rawat, and S. Oymak. Mechanics of next token prediction with self-attention. *arXiv preprint*, 2024.

[9] M. E. Sander and G. Peyré. Towards understanding the universality of transformers for next-token prediction. *arXiv preprint arXiv:2410.03011*, 2024.

[10] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[11] S. Bae et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, 2025.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, pages 5998–6008, 2017.

[13] H. Naveed et al. A comprehensive overview of large language models. *arXiv preprint*, 2023.

[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

[15] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[16] A. Chowdhery et al. PaLM: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24(240):1–113, 2023.

[17] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[18] J. Okah. Small language models (SLM): A comprehensive overview. Hugging Face, 2025.

[19] Q. Zhang et al. The rise of small language models. *IEEE Intell. Syst.*, 40(1):30–37, 2025.

[20] Z. Lu et al. Small language models: Survey, measurements, and insights. *arXiv preprint*, 2025.

[21] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[22] S. Gunasekar, Y. Zhang, J. Aneja, C. C. Teodoro Mendes, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

[23] M. Javaheripi et al. Phi-2: The surprising power of small language models. Technical report, Microsoft, 2023.

[24] M. Abdin et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[25] S. Bae, A. Fisch, H. Harutyunyan, Z. Ji, S. Kim, and T. Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *Proc. 13th Int. Conf. Learn. Represent. (ICLR)*, 2025.

[26] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[27] W. Cai et al. A survey on mixture of experts in large language models. *IEEE Trans. Knowl. Data Eng.*, 36(12):7304–7317, 2024.

[28] Y. Zhou et al. Mixture-of-experts with expert choice routing. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 35, pages 7103–7114, 2022.

[29] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, 1991.

[30] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1):5232–5270, 2022.

[31] A. Q. Jiang et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[32] DeepSeek-AI. DeepSeek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[33] S. Shen et al. Sliced recursive transformer. *arXiv preprint*, 2022.

[34] M. Elbayad, J. Gu, E. Grave, and M. Auli. Depth-adaptive transformer. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[35] A. Karpathy. char-rnn. GitHub repository, 2015. https://github.com/karpathy/char-rnn.